Detecting Network Covert Channels with Machine Learning

Zachary Riback
Rochester Institute of Technology

Alena Manchester
Rochester Institute of Technology

Daryl Johnson
Rochester Institute of Technology

Abstract

Covert channels represent an increasing threat to cybersecurity by enabling the transmission of hidden information in ways that can bypass conventional detection mechanisms. This research explores the use of Artificial Intelligence (AI) and Machine Learning (ML) to detect covert channels embedded in network communications. Specifically, we developed two main covert channel prototypes: one that manipulates the Initial Sequence Number (ISN) within the Transmission Control Protocol (TCP) and another that embeds information within the URL parameters of the HTTP protocol. Using these techniques, we sought to determine the effectiveness of AI-based approaches in identifying and distinguishing such concealed communication patterns. Our study used machine learning models to analyze traffic data, focusing on anomaly detection, feature analysis, and entropy rates. The results highlight the strengths and limitations of AI in identifying covert channels, providing critical insights into its role in enhancing network security and mitigating advanced cyber threats.

1 Introduction

With the rapid development of technology and an increasingly interconnected digital landscape, it is more important now than ever to be able to protect the integrity and confidentiality of data communications. Covert channels are used to transfer information in unconventional ways, often bypassing traditional security measures. This poses significant challenges in the cybersecurity space as these covert communications can be used for malicious purposes such as data exfiltration or unauthorized access. The ability to detect when covert channels are present can be a daunting task, but with the recent advancements made in artificial intelligence (AI), new opportunities have emerged in detecting these threats.

Artificial intelligence leverages features such as pattern recognition, anomaly detection, and predictive analysis, which are powerful tools to identify and mitigate covert channels. AI provides great potential to enhance cybersecurity defenses in both static and dynamic network environments. Our research aims to measure how well AI is able to do this over various types of covert channels. Specifically, we investigate how well our AI model is able to identify covert channels within the TCP and HTTP networking protocols.

2 Related Works

Many novel studies have been conducted on covert channels that preserve the syntax and semantics of various networking protocols [12]. This means that the channels are utilizing the existing data transmissions to carry the hidden information, making the traffic appear normal to the average observer. We have seen this done within the TCP protocol many times, such as changing IP header fields like the IP Identification Field, Initial Sequence Number (ISN), and the Acknowledge Sequence Number Field [17]. Research has also been done on storage and timing-based covert channels utilizing application layer protocols like HTTP [3].

Various methods have been explored to detect covert channels effectively within networking protocols. One study found using machine learning and data mining, combined with a hierarchical organization of frequent sets, to be an efficient and effective approach to detecting network-based covert channels [15]. Another experiment found that by embedding a Convolutional Neural Network (CNN) onto a signal receiver to monitor raw transmission signals, the model was able to not only detect when a Hardware Trojan-based (HT) covert channel was present within the communications but also predict the type of mechanism inside the transmitter with 99% accuracy [8]. Similarly, deep learning frameworks have also been used to discriminate between legitimate and illegitimate traffic within wireless networks where inter-arrival packet delays, a covert timing channel, are present [13].

Additionally, another study involved using deep learning to detect and localize covert channels by converting malicious and normal traffic into colored images. Once the images were created, a Convolutional Neural Network (CNN) was used to process the images and identify which ones contained the malicious covert channels, and it was able to do so with a 96.75% detection accuracy [1]. A survey was also done on covert channel detection and generation techniques, comparing classical approaches, modern approaches, and machine and deep learning methods. As a result of this survey, it was discovered that machine learning and deep learning detection techniques had much higher accuracy and precision than other more traditional methods of detection [2].

Other work has been done to analyze the difficulty of detecting various types of covert channels, including storage, timing, and hybrid channels. It was found that the hybrid channels were much harder to detect over just one or the other [9]. Machine learning has also been successfully used to detect network-based covert channels within Internet of Things (IoT) ecosystems with 91% accuracy and 94% precision [10]. Another study experimented with Support Vector Machine (SVM)-based frameworks to reliably detect the presence of covert timing channels. It found that the framework was very efficient at detecting covert channels, even when the size of a covert message was greatly reduced [19]. An additional study focused on detecting DNS covert channels using a long short-term memory (LSTM) model and a convolutional neural network (CNN) model. They found that the LSTM model performed much better when compared with the CNN model [4].

One study used a novel tool called CovertHunter, which uses a Large Language Model (LLM) to detect the presence of covert channels within policy configurations by comparing their natural language to actual cloud policies. This was used to help protect cloud access control policies from being tampered with and introduce vulnerabilities that would allow a malicious actor to gain unauthorized access to cloud resources [11]. This goes to show the tremendous impact that the ability to detect covert channels can have on an organization and its cybersecurity posture. Machine learning has been shown time and time again to be a very powerful tool to utilize in the detection of covert channels and has inspired us to conduct our study to determine how well it can detect channels that involve the TCP and HTTP network protocols.

3 Threat Model

When it comes to detecting covert channels, it is essential to understand the *message path* of communication the sender and receiver are utilizing as well as the type of adversary being emulated. The message path, in short, is the method by which the hidden message is transmitted, how it is embedded in the normal traffic, and how the receiver reads the hidden message. Lucena et al. discuss how two parties, referred to as Alice and Bob, can use different message paths to alter how the secret communication is facilitated [12]. For instance, Alice and Bob could be the original sender and receiver of both the *overt* and *covert* messages. Conversely, the original message could be authored by another innocent party, and Alice (the covert sender) could intercept and embed the covert message after the fact. Bob (the covert receiver) would then receive the message, strip out the embedded covert data, and send the message on its way to its final, originally intended, recipient. In other words, either one or both of Alice and Bob could be the original sender and receiver, or they could be middlemen piggybacking off of already existing communication. It is important to note that, as pointed out in [20], being a middleman does not necessarily mean being located on another physical device. The middleman could, for example, be located on the same device as the innocent sender/receiver but at a lower level in the network protocol stack.

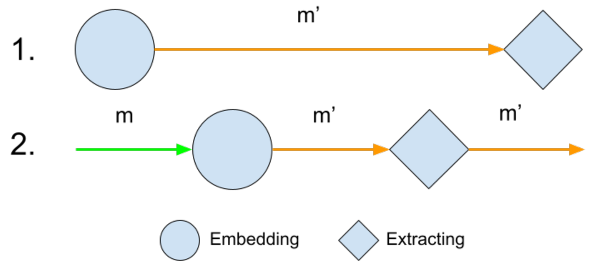

Figure 1: Message paths for two covert channels analyzed in this paper. Adapted from [12].

A *warden* is an entity seeking to detect and otherwise prevent the use of covert channel communication over a network. The different types of wardens are an important consideration for this research. Zander et al. discuss the three different types of wardens: passive, active, and malicious [20]. They are also more thoroughly described in [6]. A passive warden can only view and analyze traffic to try and detect covert channels. An active warden can slightly modify the traffic in order to try and thwart any potential secret codes. Active wardens are much more difficult to utilize in the real world, although they have been demonstrated to be effective in some cases for detecting network covert channels [7]. A malicious warden can completely alter messages or even impersonate Alice or Bob, but this is almost never seen in the real world and is not usually considered.

For this research, we act only as passive wardens. While trying to detect and mitigate covert channels, we can only passively observe the associated network traffic and analyze it for the signs and patterns of covert channels. Additionally, because of Kerckhoffs' Principle, which states that security through obscurity is never an ideal solution, **we assume the passive warden knows the type of network covert channel that may or may not be in use**.

The covert channels we analyze in this paper operate using two different message paths. One channel operates by modifying existing network traffic, akin to acting as a middleman, and the other operates by generating its own network traffic with the covert message hidden within. This is akin to Alice and Bob being both the overt and covert sender/receiver simultaneously. These message paths are illustrated in Figure 1. The covert channels we implemented for this paper and how they relate to various frameworks are described much more thoroughly in Section 4.

4 Covert Channels

4.1 TCP Initial Sequence Number Covert Channel

The first covert channel operates over TCP by encoding the secret messages within the initial sequence numbers (ISNs). Due to how ISNs are usually randomly generated by the underlying network stack, it is an ideal location to hide obfuscated secret data, and elementary implementations have been proposed and discussed several times over the years, such as in [17]. Extending from this basic idea, Rutkowska proposed NUSHU, a novel covert channel that uses control bytes and encryption to embed the covert message within the ISN [18]. This results in a more uniform and less suspicious distribution of ISNs in the network traffic, which should help to make NUSHU more imperceptible compared to other more simple solutions. Murdoch and Lewis, however, point out how this still results in a distribution of ISNs different than expected when compared to what the underlying Operating System (OS) would generate. This opens up an avenue for NUSHU and other covert channels like it to be detected [14]. They also state that NUSHU is further vulnerable because it uses DES in its encryption algorithm, but this problem is solved in our algorithm because we use AES, instead.

In order to demonstrate this weakness, we implement two TCP ISN covert channels. The first one is heavily based on [18] and is detailed in 4.1.1. The second one is designed to be much simpler and emulate some of the more elementary proposals for embedding secret data inside ISNs. It is described in Section 4.1.2.

4.1.1 Complex ISN Covert Channel

Referring back to Figure 1, this covert channel is implemented in a way that follows path 2. The covert sender and the receiver act as middlemen while being located on the same physical machine as the overt sender and receiver. On the sending machine, another application generates TCP traffic to the receiving host (or, alternatively, it generates TCP traffic to another host that the receiving host will still be able to see). The transmitter program intercepts this traffic and modifies the ISN when the connection is established. This modified ISN is encrypted and contains the secret message. The receiver views the incoming ISN, decodes it to get the secret message in plain text, and lets the message proceed to

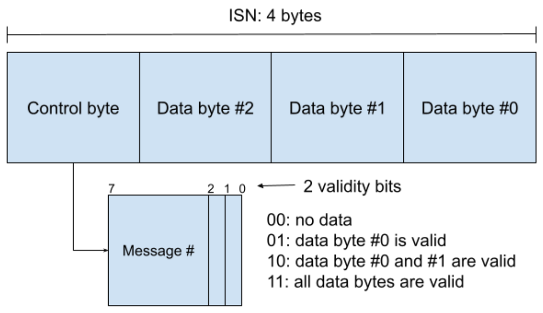

Figure 2: Format for covert ISN. Adapted from [18].

the intended recipient. Thus, the covert sender/receiver can have their secret conversation on top of the innocent, overt communication.

Each new TCP connection to the receiver allows the sender to transmit 4 bytes of information within the *covert ISN* since the sequence number in TCP is 4 bytes long. The method by which we encode the data into this value is illustrated in Figure 2. In order to create an efficient and reliable channel, the first byte is used as a control byte. It contains the message number in sequence for this packet and tells how many of the three data bytes are valid in this particular message. This allows the receiver to recognize when the sender is sending an encoded message and to verify it receives all the packets in the correct order. Consequently, each ISN can contain three bytes of the secret message.

To try and obfuscate the existence of the covert channel, the covert ISN is XORed with an encryption mask, which is illustrated in Figure 3. Firstly, the bytes "CC" are concatenated with the output of XORing the TCP source and destination port and the output of XORing the source and destination IP addresses. Every new TCP connection to a particular host will use a different source port, meaning this value will be different between subsequent TCP connections. This value is then doubled to fit the 16-byte block size of AES. The sender then uses the shared key to compute the AES output given this input value, and the first 4 bytes of the output are used as a mask with which to XOR the covert ISN. This results in the *encrypted ISN*, which is the final value sent over the wire. On the other end, the receiver performs the opposite algorithm on incoming ISNs from the sender to compute the plain text message.

In order to effectively implement this covert channel, the transmitter program needs to be able to intercept and modify the TCP sequence numbers after they are computed by the underlying operating system. When a new TCP connection is established to the receiver, the transmitter modifies the ISN with the one computed using the algorithm detailed above. The transmitter then must keep track of the difference, or offset value, between the encrypted ISN used and the ISN

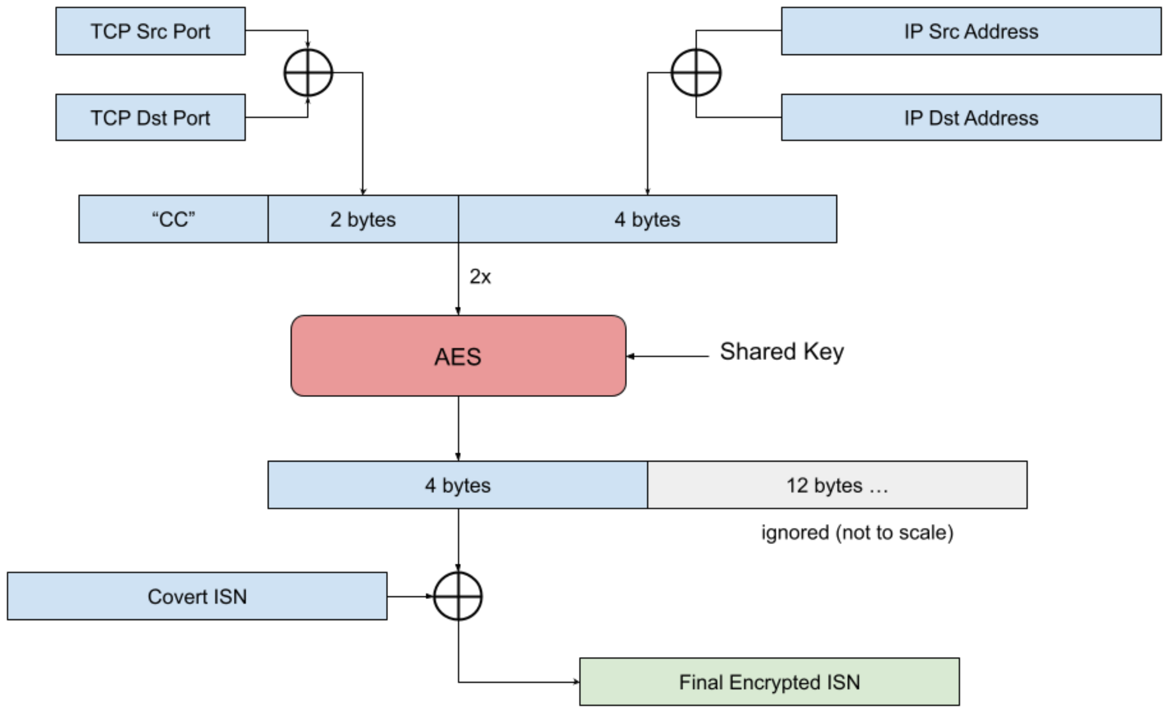

Figure 3: Algorithm for generating final encrypted ISN with the covert ISN. Adapted from [18].

originally generated by the OS. The original ISN is what the OS expects, so the transmitter must systematically modify all outgoing sequence numbers using the offset to align with the encrypted ISN. Additionally, all incoming TCP acknowledgment values from the receiver must be similarly modified to be in line with what the operating system is expecting. The receiver program does not have to do any of this since the receiver's OS is completely unaware that any of this is happening on the sending machine.

There are multiple ways for the transmitting program to achieve this effect. The most invisible is likely writing and loading a custom kernel driver to change the way the OS generates ISNs. However, this is also the most complicated and requires the highest level of permission to implement from an attacker's perspective. For this research, we implemented the algorithm as a user-space program utilizing NFQUEUE on Linux. In short, iptables rules are used to direct the proper traffic to the appropriate NetFilterQueue. An elevated running process can be created to manipulate the network packets sent to the queue. It can edit the ISN, replace it with another value, and then accept the packet to allow it to be sent to the intended recipient. The exact technical details of the implementations of covert channels are not the focus of this paper, though, so it will not be discussed further. However, the covert channel source code is available publicly. See Section 10.

4.1.2 Simple ISN Covert Channel

This covert channel operates very similarly to the more complex ISN channel previously discussed. The data is encoded in the covert ISN in the same way. The only difference is in the algorithm shown in Figure 3. Instead of this procedure, the covert ISN is simply XORed with a constant 4-byte key shared between the sender and receiver to generate the encrypted ISN.

4.2 HTTP GET Request Parameters Covert Channel

The second covert channel that is implemented operates by generating HTTP GET requests with the encoded information stored in the parameter data. As described by Lucena et al., modifying parameter data is not semantically preserving [12]. Therefore, this covert channel cannot function by modifying already existing HTTP traffic like the previously discussed channel modifies TCP traffic. The transmitting program must create the HTTP traffic itself. This inherently makes the channel easier to detect because all an attacker has to do is see if their is extra HTTP traffic on the network that should not be present. The task for the sender is to hide their covert message-carrying HTTP traffic among other legitimate traffic on the network. This covert channel follows message path 1

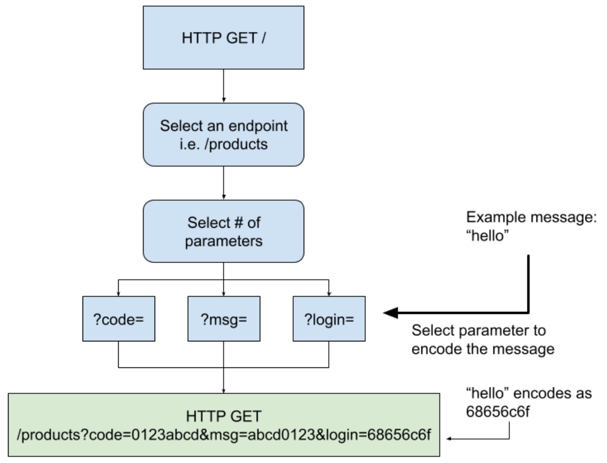
HTTP GET /
Select an endpoint i.e. /products
Select # of parameters
?code=
?msg=
?login=
Example message: "hello"
Select parameter to encode the message
"hello" encodes as 68656c6f
HTTP GET /products?code=0123abcd&msg=abcd0123&login=68656c6f

Figure 4: HTTP covert channel operation

Role	IP Address	Kernel
Main Sender	10.30.0.1	Linux 6.8.0-48-generic
Legacy Sender	10.30.0.3	Linux 2.4.21-50
Main Receiver	10.30.0.2	Linux 6.8.0-48-generic

Table 1: Test environment network information

as shown in Figure 1. The sender and receiver are the authors of both the overt and covert messages.

The operation of our designed HTTP covert channel is illustrated in Figure 4. First, the sender picks an endpoint to which to send the HTTP GET request. Note that the receiver does not necessarily need to be running the web server. Just as long as it can see the HTTP request the sender is making, it will be able to decode the message. The sender also decides on a number of parameters to use, from one to three. In the case of the diagram, three parameters are used, and they are randomly selected from a predefined set. Next, the message is encoded in hex and put as the value for one of the parameters. The other parameters are filled with random data as appropriate. On the receiver side, it views all HTTP GET requests from the sender and attempts to decode each parameter value. If it can successfully decode the parameter into a valid ASCII string, then it is considered part of the covert message. If none of the parameters can be properly decoded, then that HTTP packet is considered to not have any encoded data. Realistic use of this covert channel allows the sender to skip any number of valid HTTP requests in between encoding the secret data. Therefore, the transmitting program could run concurrently with other applications or user actions that generate HTTP GET requests.

5 Data Collection

In order to train machine learning models on the network traffic from these covert channels, we aimed to have 1,000 packet capture samples for each case. That means for each covert channel, generating 1,000 packet captures while the covert channel is running and 1,000 samples while it is not for comparison. Together, these 2,000 samples form the total dataset for that covert channel. With this, we can train the machine learning model to differentiate between the two cases. The test environment is briefly detailed in Table 1, and the mechanisms by which the data is collected are outlined per covert channel in this section. Additionally, all the datasets collected and used in this research are publicly available. See 10.

5.1 TCP ISN Covert Channels

Multiple datasets were collected for the TCP covert channels. The first dataset pertains to the complex ISN channel outlined in Section 4.1.1. To collect one sample of the active covert channel, a network packet capture is started on the main sender host in the test environment. Then, the transmitter program is started, officially making the covert channel active. Over the course of the next 60 seconds, 20 TCP connections are initiated and terminated by the sender to the receiver on a predetermined port on which the receiver is listening. Each of these 20 connections represents three bytes of the secret message being transmitted. For each iteration, a random message written in English is chosen. In order to generate a network sample where the covert channel is inactive, an identical procedure is followed except without enabling the covert channel.

Another dataset was captured for the simple ISN channel, as described in Section 4.1.2. An identical procedure to the one just described was followed except with the modified transmitter program with the simple encryption algorithm.

Finally, a third dataset was captured using the legacy sender with Linux kernel version 2.4.21-50. This dataset is just 1,000 samples, each containing 20 TCP connections initiated from the legacy sender. This gives us a dataset of TCP ISNs generated from the 2.4 Linux kernel, which we would expect to be distributed differently than those generated from a more modern kernel version. This is because advances have been made over the years to make generated ISNs more cryptographically random. Version 2.4 was the Linux kernel version in mind while NUSHU [18] was being developed and also when Murdoch and Lewis gave criticism to NUSHU, saying it would not generate an ISN distribution similar to the distribution generated by the underlying operating system [14]. It was stated that because of how the 2.4 kernel generates ISNs, NUSHU was weak to various tests for covert channels. However, the researchers do not explicitly demonstrate this supposed truth. This dataset represents 1,000 samples where no covert channel is active, and we can match it with 1,000 samples where our NUSHU-based covert channel was active to create a complete dataset on which to train.

5.2 HTTP GET Request Parameters Covert Channel

Similar datasets were collected for the HTTP covert channel. In this case, one sample is one packet capture containing 12 HTTP GET requests from the main sender to the main receiver in the test environment. In a capture where the covert channel was active, a random three subsequent HTTP requests will have a secret message encoded in one of their parameters. Any other parameters in those requests will still hold random data. The secret messages are all fairly short, written in English, and randomly chosen for each packet. Note that even in a capture where the covert channel is active, 9 of the 12 total HTTP requests will still be filled with random data and will have no encoded message. A sample where the covert channel is not active will have all 12 HTTP GET requests with parameters filled with random data.

6 Machine Learning

6.1 Feature Extraction

Before the data can be fed into the deep learning model, features need to be extracted. As mentioned before, we assume the passive warden knows the covert channels that may or may not be used on the network at any given time. This means that the features extracted from the data and fed to the model will be very specific to the covert channel we aim to detect. To be specific, only the ISNs will be extracted from the packet captures to detect the TCP ISN covert channels. The input to the model will be a list of ISNs corresponding to each ISN generated in the packet capture. Conversely, for the HTTP covert channel, the model will train based on the calculated Shannon entropy of the byte string obtained by concatenating the values of all the HTTP GET parameters for a given packet. Entropy is a good metric to examine in this case since it is a simple value that can reveal whether or not data is randomly generated, which is exactly what the model will attempt to determine. A summary of the extracted features for each covert channel dataset is given in Table 2. Note that no active samples were captured for the legacy TCP channel. This data will be matched with the active data from the complex TCP channel to create a full dataset on which to train.

6.2 Model Training

After extracting the features and creating the appropriate numpy structures, they are saved to object files, uploaded to Google Drive, and then loaded into a Jupyter Notebook. A 1D convolutional neural network was implemented to create a trained model from the data. With our 2,000 samples for each covert channel test, we utilize a train/test split of 90/10, meaning 90% of the data is used for training while 10% is used to test the accuracy of the resulting model. The machine learning code is written in Python and utilizes the Keras [5] and Scikit-learn [16] libraries. The model uses multiple convolutional blocks implementing batch normalization, pooling, and dropout layers. There are also bidirectional LSTM and fully connected layers. The model was designed by gradually increasing complexity until the approximate desired training time and positive results were achieved. The results of all machine learning tests are summarized in Table 3.

Because this experiment deals with balanced classes (i.e., there are an equal number of samples of active and inactive covert channels), accuracy is the only metric that is analyzed. In fact, due to the nature of the data collected and the experiment conducted, it can be expected that the accuracy, precision, and recall values will be very similar to each other for any resulting model. The full machine learning code is also available publicly. See Section 10.

7 Findings

From the results shown in Table 3, we see how various levels of success were achieved when detecting covert channels using deep learning. Firstly, a 100% accuracy figure for the legacy TCP dataset illustrates the validity of the point raised by Murdoch and Lewis [14]. They stated that differences in the way NUSHU [18] generates ISNs compared to how the Linux Kernel version 2.4 generates them would allow the presence of the covert channel to be detected. Given our results, this can be solidly confirmed. Additionally, an accuracy rate hovering around 50% for our complex TCP covert channel shows how this weakness has been patched in the more modern Linux kernel. Despite our best efforts, we were not able to train a model that could accurately distinguish between ISNs generated by our NUSHU-based covert channel and the native Linux OS running kernel 6.8. This means that the ISN generation in the more recent version has been updated to be much more random and more closely resemble the random distribution we would expect from AES implemented as part of the encryption scheme for our complex TCP algorithm. Additionally, our simple TCP algorithm yielded an accuracy of 100%. This is expected and acts as a pseudo-control for our experiment. Clearly, the covert channel is highly vulnerable if the data is not encrypted with a sufficiently complex algorithm.

Finally, the HTTP covert channel results reveal a strong accuracy value considering the challenges the model had in learning from this data. As described in Section 4.2, only a small portion of each capture is actually used to transmit the secret message when the covert channel is active. In an HTTP GET request packet with multiple parameters, only one of them will contain the secret message, and the others will still

Covert Channel	Feature	# Active Samples	# Inactive Samples	Data points/sample
Complex TCP	ISNs	1000	1000	20
Simple TCP	ISNs	1000	1000	20
Legacy TCP	ISNs	-	1000	20
HTTP	Parameter Entropy	1000	1000	12

Table 2: Summary of data and features extracted

Covert Channel	Avg. Accuracy (%)	Avg. Time (s)
Complex TCP	49.55	43.70
Simple TCP	100	46.56
Legacy TCP	100	29.55
HTTP	75.20	28.48

Table 3: Summary of covert channel machine learning results from 10 tests

hold random data. Additionally, the model only works with the entropy values of the HTTP parameter data and nothing else. We would expect there to be marginally less entropy with the covert channel data compared to the non-covert channel data; encoded (not encrypted) English text will be less random and possess more patterns than randomly generated data. This is the case, as the average entropy of all HTTP GET request parameter data when the covert channel was active (including the majority of packets in each capture where no data was encoded) is about 3.636 bits, while the average entropy when the channel is not in use is 3.719 bits. The challenge for the machine learning model is whether it can detect this difference well enough on a sample-by-sample basis to make accurate classifications. A result of 75.20% accuracy shows that it can and that more data and model fine-tuning could result in even higher accuracy values. This result also illustrates how a covert channel is likely vulnerable to this detection method if a proper encryption algorithm is not used. Because English text has much less entropy than that of random data, entropy will always be a metric that could be used to detect covert channels. This result also shows that even if not all the packets are used in the covert channel, that is not protection enough from AI detection methods given enough data.

8 Conclusion

In this paper, we present our extensive covert channel implementations, data collection and cleaning methods, machine learning models, and the results of training on thousands of samples. We confirm the idea asserted in one paper [14] and demonstrate how, with newer updates to the Linux kernel, those concerns are no longer founded. We also demonstrate how covert channels that do not utilize encryption to obfuscate the covert data are fundamentally vulnerable to AI-based attacks because the distribution of covert data will be demonstrably different than that of normal data. If the data generated by the covert transmitter is too dissimilar to the data it is trying to mimic, that is a weakness that might be difficult for a human to detect but could be trivial for an AI model. Overall, this research exemplifies how machine learning must be considered as a tool for a passive warden before a proposed covert channel is declared secure.

9 Future Work

Future work in this space should certainly involve testing more network covert channel types. There are likely hundreds, if not thousands, of proposed covert channels out there (some implemented and some not) that have never had their security tested with a machine learning model. Performing these tests would go a long way in verifying if these channels are as secure and imperceptible as their authors usually claim. It would certainly be valuable to examine a timing-based covert channel, which is a type of channel that was not studied in this paper. Additionally, in this paper, only one type of neural network was utilized, and not much fine-tuning or testing was done on the model. In the future, more experimentation could be done to find the best model for detecting network covert channels.

We also assumed that the passive warden knew the exact type of covert channel that was active. It would be worthwhile to demonstrate, perhaps using the data collected during this research, how successful a machine learning model could be in detecting covert channels when it is unknown the exact relevant features that should be extracted from the network captures.

10 Availability

All the raw datasets, cleaned datasets, covert channel code, data collection code, processing code, and machine learning code are publicly available at https://github.com/zriback/CC-Detection. See the README.md for more information on the included files.

References

[1] AL-EIDI, S., DARWISH, O., HUSARI, G., CHEN, Y., AND ELKHODR, M. Convolutional neural network structure to detect and localize ctc using image processing. In *2022 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS)* (2022), pp. 1–7.

[2] AL-Khulaidi, N. A., Zahary, A. T., Hazaa, M. A., and Nasser, A. A. Covert channel detection and generation techniques: A survey. In *2023 3rd International Conference on Emerging Smart Technologies and Applications (eSmarTA)* (2023), pp. 01–09.

[3] Brown, E., Yuan, B., Johnson, D., and Lutz, P. Covert channels in the http network protocol: Channel characterization and detecting man-in-the-middle attacks. *Journal of Information Warfare 9*, 3 (2010), 26–38.

[4] Chen, S., Lang, B., Liu, H., Li, D., and Gao, C. Dns covert channel detection method using the lstm model. *Computers Security 104* (2021), 102095.

[5] Chollet, F., et al. Keras. https://keras.io, 2015.

[6] Craver, S. On public-key steganography in the presence of an active warden. In *Information Hiding* (Berlin, Heidelberg, 1998), D. Aucsmith, Ed., Springer Berlin Heidelberg, pp. 355–368.

[7] Dakhane, D. M., and Deshmukh, P. R. Active warden for tcp sequence number base covert channel. In *2015 International Conference on Pervasive Computing (ICPC)* (2015), pp. 1–5.

[8] Díaz-Rizo, A. R., Abdelazim, A. E., Aboushady, H., and Stratigopoulos, H.-G. Covert communication channels based on hardware trojans: Open-source dataset and ai-based detection. In *2024 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)* (2024), pp. 101–106.

[9] Goher, S. Z., Javed, B., and Saqib, N. A. Covert channel detection: A survey based analysis. In *High Capacity Optical Networks and Emerging/Enabling Technologies* (2012), pp. 057–065.

[10] Guarascio, M., Zuppelli, M., Cassavia, N., Manco, G., and Caviglione, L. Detection of network covert channels in iot ecosystems using machine learning. In *ITASEC* (2022), pp. 102–113.

[11] Karmarkar, H., Joshi, V., and Venkatesh, R. Detecting covert channels in cloud access control policies using large language models. In *2024 IEEE International Conference on Cyber Security and Resilience (CSR)* (2024), pp. 241–246.

[12] Lucena, N. B., Pease, J., Yadollahpour, P., and Chapin, S. J. Syntax and semantics-preserving application-layer protocol steganography. In *Information Hiding* (Berlin, Heidelberg, 2005), J. Fridrich, Ed., Springer Berlin Heidelberg, pp. 164–179.

[13] Massimi, F., and Benedetto, F. Artificial intelligence-based hidden communications detection in wireless networks. In *2023 46th International Conference on Telecommunications and Signal Processing (TSP)* (2023), pp. 197–203.

[14] Murdoch, S. J., and Lewis, S. Embedding covert channels into tcp/ip. In *Information Hiding* (Berlin, Heidelberg, 2005), M. Barni, J. Herrera-Joancomartí, S. Katzenbeisser, and F. Pérez-González, Eds., Springer Berlin Heidelberg, pp. 247–261.

[15] Nowakowski, P., Zórawski, P., Cabaj, K., and Mazurczyk, W. Detecting network covert channels using machine learning, data mining and hierarchical organisation of frequent sets. *J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl. 12*, 1 (2021), 20–43.

[16] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[17] Rowland, C. H. Covert channels in the tcp/ip protocol suite. *First Monday* (1997).

[18] Rutkowska, J. The implementation of passive covert channels in the linux kernel. In *Proc. Chaos Communication Congress* (2004).

[19] Shrestha, P. L., Hempel, M., Rezaei, F., and Sharif, H. A support vector machine-based framework for detection of covert timing channels. *IEEE Transactions on Dependable and Secure Computing 13*, 2 (2016), 274–283.

[20] Zander, S., Armitage, G., and Branch, P. A survey of covert channels and countermeasures in computer network protocols. *IEEE Communications Surveys Tutorials 9*, 3 (2007), 44–57.